John von Neumann Institute for Computing

Parallel Algorithm for Nonlinearly Unconstrained Optimization Based in Parametric Trees

I. Pardines, D.E. Singh, F.F. Rivera

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 253-260, 2006.

http://www.fz-juelich.de/nic-series/volume33

Parallel Algorithm for Nonlinearly Unconstrained Optimization based in Parametric Trees

I. Pardines[a], D. E. Singh[b] and F. F. Rivera[c]

[a]Departamento de Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain

[b]Departamento de Informática, Universidad Carlos III de Madrid, 28911 Leganés, Spain

[c]Departamento de Electrónica y Computación, Universidad de Santiago de Compostela, 15782 Santiago de Compostela, Spain

Algorithms for solving unconstrained nonlinear optimization problems are computational expensive. In each iteration of these iterative methods, a search direction and a stepsize along this direction are generated. Both computations influence the number of iterations and the number of function evaluations required for the whole process. A reduction in the total number of iterations and function evaluations implies a significant improvement in the performance of the algorithm. In this context, multi-step parallel algorithms based on the execution of a sequence of trees are proposed in this paper. Different damping parameters are used to define the branches of the tree. Each branch computes a different stepsize. The performance of the parallel algorithm depends on the number of branches and the depth of the tree, which can be parameterized and controlled by the user. Results on a set of test problems to validate the efficiency of our proposals are shown.

1. Introduction

Unconstrained nonlinear optimization problems arise in many applications in science and engineering. Formally, they can be expressed as,

$$\begin{array}{ll} min & f(x): \quad I\!R^n \rightarrow I\!R, \\ x \in I\!R^n & \end{array} \tag{1}$$

where the objective function $f(x)$ is assumed to be at least twice continuously differentiable. These problems are often expensive to solve in terms of computation resources. The reason can be the size of the problem and/or the complexity of the objective function. Moreover, sometimes when $f(x)$ is complex, the gradient of $f(x)$ is not available analytically. In this cases, finite or central differences are used to approach the gradient, which implies more function evaluations. In this work, we are concerned with parallel numerical methods for solving these type of problems.

A well known solution to deal with these problems is to use quasi-Newton methods. The parallelization of these methods have been considered by several researches in the past decades. There are two different ways to deal with this problem, one is to reduce the computational time per iteration, parallelizing the most costly algebra routines, and the other is to reduce the total number of iterations of the method. The authors have been working in the context of the first approach during the last past years [10] [11]. In these papers a parallel implementation of both, the Hessian matrix update and the computation of the search direction were proposed.

Some works have been devoted to other problems where the function and the gradient evaluations require high computational times, such as a function evaluation that involves the solution of a system of differential equations or a large scale simulation. For instance, the parallel strategy to compute

the gradient at each iteration is straightforward using finite differences. Formally, this computation implies $n + 1$ function evaluations, being n the number of variables, as

$$\nabla f(x)_i = \frac{f(x + h_i e_i) - f(x)}{h_i}, \tag{2}$$

where h_i is a small stepsize, and e_i denotes the i^{th} unit vector. In this case, if P processors are available, each processor can compute $\lceil n/P \rceil$ evaluations of the function, but during the evaluation of $f(x)$ in the line search, $P - 1$ processors are idle. To avoid this drawback, Schnabel proposed, in [13], a speculative technique to evaluate $max\{P-1, n\}$ components of the gradient at the updated solution point at the same time that this solution point is computed. Furthermore, Byrd, Schnabel and Shultz [2] suggest to speculatively compute the gradient and some portion of the Hessian matrix, and to incorporate this partial Hessian information into the BFGS update.

Finally, there is other ways to approach the parallelization of quasi-Newton methods that has been suggested by Lootsma [7], van Laarhoven [6], and more recently, by Phua [12]. The idea is to compute, at each iteration, P independent search directions (multi-search methods) or P different steplengths along a given search direction (multi-step methods). For massive parallel systems these two approaches can be combined to achieve high performance. The computational time is reduced by looking for different solution points at each iteration. There are two reasons for this reduction; first, the decrease in the number of function evaluations in the line search procedure and, second, a fast convergence of the method.

In this paper, a description of quasi-Newton methods, specifically the BFGS method, is presented in Section 2. Our proposals based on a parametric tree to parallelize a multi-step BFGS method are described in Section 3. Results obtained by our parallel methods in a cluster of PCs are shown in Section 4. Finally, Section 5 summarizes the main conclusions of this work.

2. Quasi-Newton methods

The success of Newton-type methods is the use of the curvature information provided by the Hessian matrix, which allows a quadratic model of the objective function. However, quasi-Newton methods are based on the idea that an approximation to the curvature of a nonlinear function can be computed without explicitly forming the Hessian matrix. This information is collected as the iterations of a descent method proceed, using the behavior of the objective function and its gradient. Therefore, these methods use an approximation to the Hessian matrix instead of the Hessian itself. At each iteration of the method, when a new solution point is considered, the approximation of the Hessian matrix has to be updated. One important feature of quasi-Newton methods is the choice of the Hessian matrix approximation. There are different update formulas [5]. One of the most commonly used to solve unconstrained nonlinear optimization problems is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update.

The BFGS method works with the Hessian matrix approximation, however, to solve the required system of equations is more efficient to use the inverse of the Hessian matrix approximation. The algorithm that uses this matrix is called the inverse BFGS method, and it is the one that has been considered in this work.

Given an initial point x_k, a gradient g_k and a Hessian matrix approximation H_k, the inverse BFGS method executes a set of stages at the k-th iteration. Firstly, a direction of search $d_k = -H_k^{-1} g_k$ is computed. After that, an optimal stepsize α_k, along this search direction from x_k is found. Therefore, the next iteration point is given by $x_{k+1} = x_k + \alpha_k d_k$. Finally, the approximated Hessian matrix is updated by the correction matrix D_k, in such a way that $H_{k+1} = H_k + D_k$.

A detailed profiling analysis shows that one of the most expensive stages of the inverse BFGS method is the computation of the steplength along a search direction in every iteration. Both, the steplength and the search direction influence the number of iterations required for the whole optimization process. The search direction can be modified using a different quasi-Newton update. However, as the BFGS update presents the best behavior in most cases, we just propose to modify the computation of the steplength. The line search procedure is summarized in detail in the next section.

2.1. The line search procedure

The line search procedure computes the steplength at each iteration. Computing the steplength α_k at the k-th iteration implies the solution of the following one-dimensional optimization problem [8]:

$$f(x_k + \alpha_k d_k) = min \quad f(x_k + \theta d_k), \quad with \quad 0 < \theta \leq \alpha_{max}. \tag{3}$$

The selected steplength has to verify the Wolfe's conditions,

$$f(x_k + \alpha_k d_k) - f(x_k) \leq \mu g_k^T d_k, \tag{4}$$

$$\nabla f(x_k + \alpha_k d_k) \geq \eta g_k^T d_k, \tag{5}$$

being μ, $\eta \in (0, 1)$.

The best methods to solve this type of problems are the so called safeguarded procedures. These methods can be considered combinations of the bisection and the linear interpolation methods. Their efficiency is based on the use of information from the objective function. Polynomials are frequently used to interpolate $f(x_k + \theta d_k)$. If the approximation is inaccurate the procedure may diverge, so an interval of uncertainty $[a, b]$ is maintained as safeguard.

The line search procedure finds a sequence of improving estimates of a minimizer of $f(x_k+\theta d_k)$ in the interval $(0, \alpha_{max}]$. In our proposal, safeguarded cubic interpolation has been used to compute the sequence of estimates. The procedure computes an interval of uncertainty and the two "best" points of the objective function are obtained to be used as interpolation points. Each time an estimate is computed, for being selected it has to verify the Wolfe's conditions. If these conditions fail the interval is changed and a new estimate is searched.

The selection of the initial step is critical to achieve a global convergence of the method. At the k-th iteration, in the first evaluation of function f, a potentially small steplength $\alpha_{k0} = min(1, \gamma)$, where $\gamma = r(1 + ||x_k||_1)/||d_k||_1$, is used. The parameter r, called damping parameter, limits the change in the current solution point x_k during the line search. Therefore, evaluations of the objective function at meaningless points are prevented. A suitable damping parameter can be fixed according to the particular features of the problem. For example, low values, such as 0.1 or 0.01, may be helpful when the function varies rapidly.

We are interested in multi-step methods that compute, at every iteration, several steplengths. We suggest to initialize different values of α using different damping parameters, and establishing several parallel quasi-Newton executions, each one starting from a different initial step. The parallel strategies are described in the next section.

3. Parallel tree techniques

A set of parallel implementations of a multi-step quasi-Newton method based on the parallel execution of a tree structure is proposed. Each branch of the tree is characterized by a different selection of the damping parameter at each node. A node is related with a different iteration of the

method, so each branch runs a quasi-Newton method in a processor of the parallel system with a different sequence of damping parameters. The way in which the damping parameters are defined, and the criterium to select the initial branches of the next tree, each time a checkup is made, are the differences between the parallel methods. Following, a description of these strategies is presented, highlighting their benefits and drawbacks.

3.1. Method 1

Since this method is based on the execution of the computations associated to a tree structure, it is necessary to describe how that tree is built. The number of nodes per branch (β) and the number of branches characterize the tree. β is an externally fixed parameter, and the number of branches of the tree corresponds to the number of damping parameters $n(r)$. Each branch consists of a set of nodes each one with the same value of the damping parameter. This implementation of the tree is justified by the empirical observation that the best solution is achieved selecting the same damping values during a certain number of concurrent iterations. In the example of Figure 1(a), five values of r are considered and $\beta = 2$. Therefore, five branches are generated after pairs of iterations.

Once the tree has been defined, in the first iteration, the algorithm starts the execution of the computational load associated to this tree. Every branch runs a quasi-Newton method in a different processor with the associated values of damping. Therefore, as many processors as the number of branches are needed. This tree remains active for β iterations. As soon as the algorithm completes the computations associated to the tree, the branch that gives the best solution at the moment is checked. The criterium to select the solution is based on the minimum value of the objective function. Each time a checkup is performed to select the best branch, the current solution point associated to this branch has to be broadcasted to the rest of the processors of the system. Therefore, a message is needed after each β iterations. The arrows in Figure 1 represent *Allreduce* operations to decide which branch achieve the best solution.

When the processors receive the current solution point, a new tree is performed, starting from this point. The process continues until the quasi-Newton method converges.

Our heuristic presents two degrees of freedom: the width and the depth of the tree. The number of branches is characterized by the width of the tree, and it depends on the amount of different damping parameters considered. The depth of the tree is defined as the number of iterations between checkups. The performance of the method is dependent on the parameters that define the tree. However, in practice the behavior of the method is unpredictable, because the method presents a drawback. That is, sometimes the branch that is selected as the best in a certain iteration is not the branch that converges in the minimum number of iterations. For this reason, the method do not always achieve the best solution. So, a second method is proposed as an alternative.

3.2. Method 2

This method is used to deal with the main drawback of Method 1. The idea is to select more than one branch each time the checkup is performed. Therefore the m best branches are kept alive. The value of m depends on the number of processors of the parallel system. In this way, after the execution of the first tree, m solution points of the quasi-Newton method are available. These points are used as starting points to perform the branches of new m trees. These trees have again as many branches as damping parameters are used, as can be seen in Figure 1(b). The process continues until the solution of the optimization problem have been achieved.

The disadvantage of this method respect to Method 1 is that more computations are needed. In this case, $P = m \cdot n(r)$ processors are required, instead of the $n(r)$ processors of Method 1. Note that, from a parallel point of view, in the computation of the first tree $(m-1) \cdot n(r)$ processors are

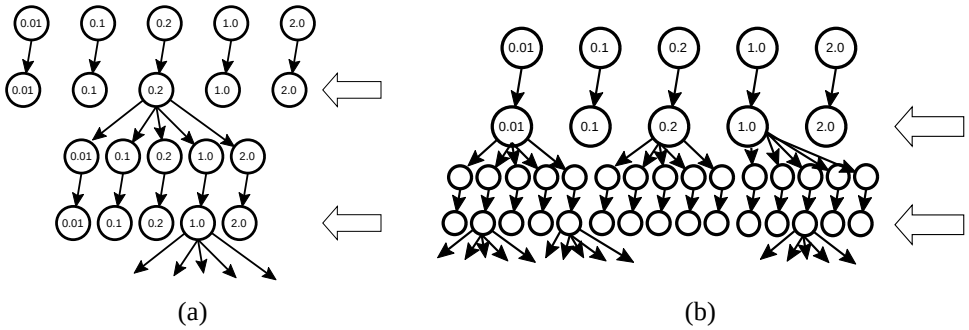

Figure 1. (a) An example of the application of Method 1. (b) An example of application of Method 2 with $m = 3$. For both cases the set of dampings=$\{0.01, 0.1, 0.2, 1.0, 2.0\}$ and $\beta = 2$.

idle.

3.3. Method 3

Experimental results show that the selection of damping parameters is important in order to achieve high performance in the execution of the parallel algorithm. The previous two methods use a set of unchangeable damping parameters given by the user. However, a new degree of freedom can be added, if the values of the damping parameters are under the user control. Depending on the type of the problem, the user can decide to use a set of random damping parameters when the behavior of the problem is unknown (Method 1 and Method 2) or to start with a suitable damping and try to find a better value. In this last case, we propose to modify the chosen damping a certain percentage x. If the number of variations over this damping value is v, a tree with $2v + 1$ branches is generated. The proposed damping parameters for these branches are:

$$r_i - v \cdot x\%, \cdots, r_i - x\%, r_i, r_i + x\%, \cdots, r_i + v \cdot x\% \tag{6}$$

Therefore, the number of processors in this method is $P = m \cdot (2v + 1)$. The user decides, based on the size of the parallel system, to increase m or v. In both cases, the controlled parameter is the width of the tree.

Method 3 can be used after an initial fast execution of Method 1. In this way, Method 1 can be executed with a selected set of dampings during some few iterations, then, the damping parameter that lead to the best solution in Method 1 is used as the starting damping for Method 3. This method will refine the damping parameter by controlled variations, improving the performance.

4. Experimental results

The parallel strategies were implemented using MPI in a cluster of PCs. Our benchmark consists of three well known optimization problems from the CUTE library [1] called “Power”, “Penalty1” and “Powellsg”, that come from three different and representative applications. The results are compared with the solutions achieved by the optimization software MINOS [9]. The percentage of reduction in the total number of iterations ($Iter$) and in the number of function evaluations ($Efun$) achieved by Method 1 respect to MINOS are shown in Figure 2. A set of five different damping parameters, and values of β from 2 to 10 are used to solve these problems in a parallel system of 5 processors.

The efficiency of Method 1 depends on the type of the problem. The highest performance is achieved for the ”Power” problem, where a maximum speedup of 4.15 is obtained. For the other two problems low values of β implies discrete results. However, as the depth of the tree increases, the performance improves. Note that, even a small reduction in the number of function evaluations is important, because it implies a minor cost in the computation of the steplength. Both reductions are more significant when there is no analytic gradient available, and it has to be computed by finite differences.

It can be observed that for some depths of the tree, poor improvements are achieved. The reason is that according to the mentioned drawback of Method 1, in some cases the best branch selected in each iteration is not the one that produces a quicker convergence.

The same benchmark and the same set of damping parameters have been used to study Method 2. In this case, the $m = 10$ branches with the lowest value of the objective function are selected each checkup. So, the number of processors required in this case is $P = 50$.The results of Method 2 versus Method 1 are shown in Figure 3 for problems with $n = 20$. Results for the “Power” problem are not shown because they are the same for both methods. The reason is that this is a well conditioned problem for which the branch selected as the best one in Method 1 leads always to a fast convergence. For the other problems, the greater improvement is achieved in the reduction of the number of iterations, and specifically, in the “Powellsg” problem, which has the worst results with Method 1. Also, it is interesting to highlight that for some depths of the tree the solution achieved by Method 1 fits with the solution of Method 2.

A comparative of the three methods is presented in Figure 4. Method 3 has been applied keeping only one branch alive each checkup in order to use less processors, explicitly $P = 7$ is used. The damping parameter that leads to the best value of the objective function in Method 1 is used. The selection of the parameters x and v has been made by means of empirical experiments.

In this figure, the different behavior of the two problems is stood out. From the ‘’Penalty1” problem, improvements between 40% and 50% in both the number of iterations and the number of function evaluations are achieved. Similar results were obtained for the three methods with values of β over 4 iterations. Method 3 shows a good behavior, very close to Method 2. Therefore, as the number of processors needed by this method is much smaller than the one used in Method 2, Method 3 seems the best option to optimize problem “Penalty1”. The behavior with the depth of the tree is much irregular for the problem “Powellsg”. In this case, Method 2 presents the best

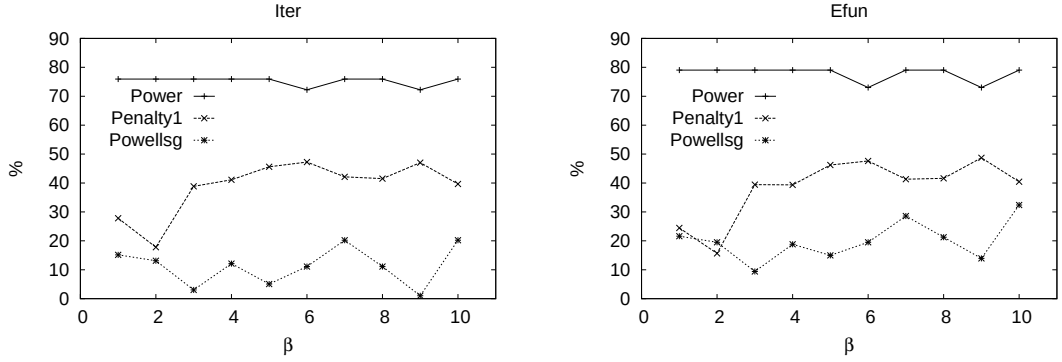

Figure 2. Percentage of improvement obtained by Method 1 for problems with 20 variables in a parallel system of 5 processors.

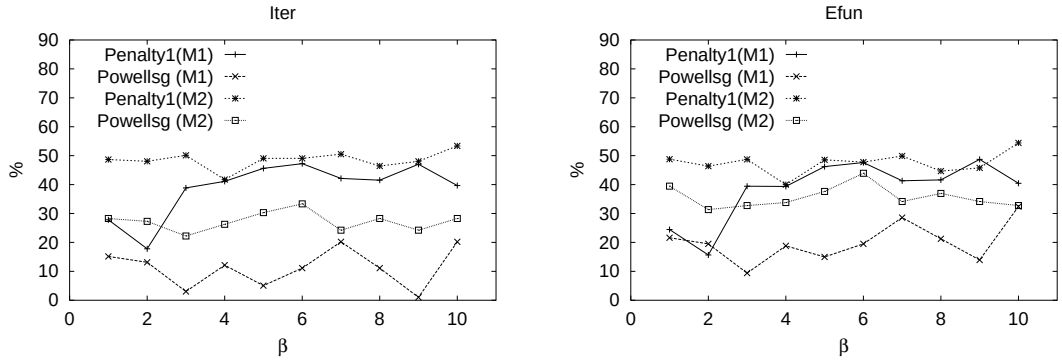

Figure 3. Improvement obtained by Method 2 versus Method 1 for problems of size $n = 20$. The number of processors is P=5 in Method 1 and P=50 in Method 2.

performance in most cases. However, when the depth of the tree is equal to 4, Method 3 achieved the best improvement, over 40%. This example stands out how the suitable depth of the tree depends on the type of the problem and on the method.

5. Conclusions

In this work different strategies to parallelize quasi-Newton methods are described. Three parallel multi-step algorithms based on the use of parametric trees are proposed. Our objective is to improve the performance of quasi-Newton methods by reducing the number of iterations and the number of function evaluations. The proposed heuristics present two degrees of freedom, the width and the depth of the tree. The width is related to the number of damping parameters, and the depth is defined as the number of iterations between consecutive checkups. Method 3 adds the set of damping parameters as a new way of controlling the width of the tree.

Depending on the type of the optimization problem, the best solution is obtained for a tree with a certain width and depth. However, we can conclude that choosing a suitable set of damping parameters and an appropriate depth of the tree, an improvement in the performance of the quasi-Newton methods is obtained. Moreover, Method 3 turns out as the best alternative when the behavior of the problem is known, since achieves good performances using small parallel systems.

Acknowledgments

This work has been supported by CICYT under projects TIC 2002/750, TIN 2004-02156 and TIN 2004-07797-CO2.

References

[1] I. Bongartz, A. R. Conn, N. Gould and P. L. Toint, CUTE: Constrained and unconstrained testing environment, ACM Transactions on Mathematical Software 21 1 (1995) 123-160.

[2] R. H. Byrd, R. B. Schnabel and G. A. Shultz, Parallel Quasi-Newton methods for unconstrained optimization, Mathematical Programming 42 (1988) 273-306.

[3] J. A. Ford and I.A. Moghrabi, Minimum Curvature Multistep Quasi-Newton Methods, Computers and Mathematics with Applications 31 (1996) 179-186.

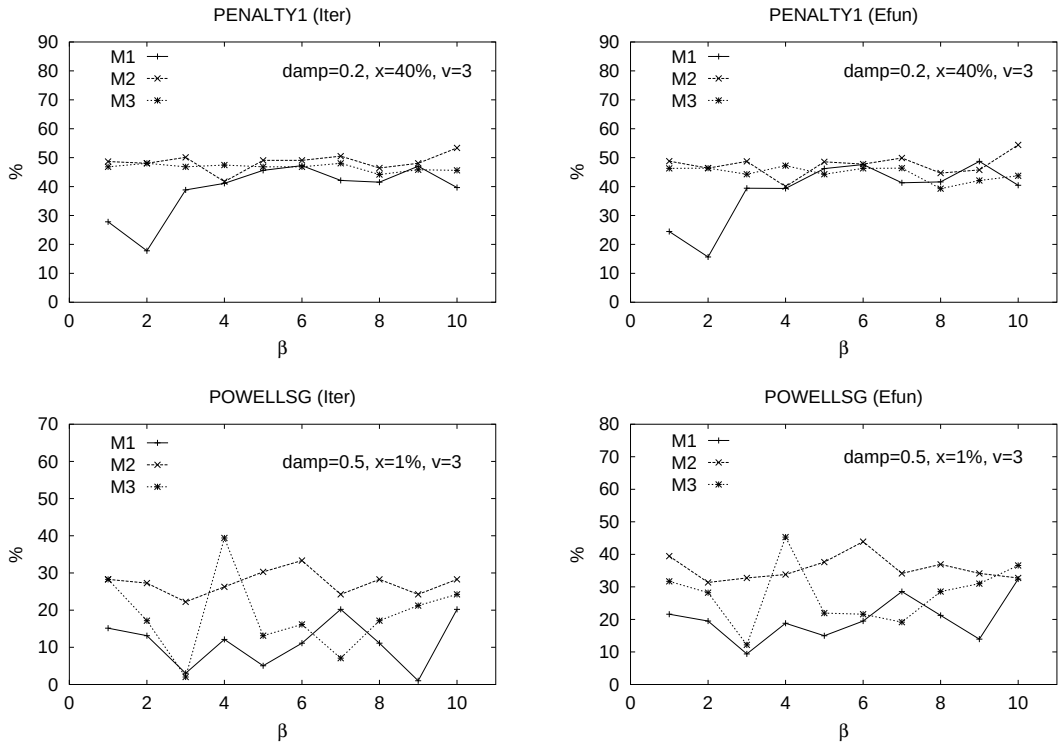

Figure 4. Comparative of the three methods for problems of size $n = 20$.The number of processors in Method 3 is P=7.

[4] J. A. Ford, Implicit Updates in Multi-step Quasi-Newton Methods, Computers and Mathematics with Applications 42 (2001) 1083-1091.

[5] Philip E. Gill, Walter Murray and Margaret H. Wright, Practical Optimization (Academic Press, London, 1981).

[6] P. J. M. van Laarhoven, Parallel variable metric methods for unconstrained optimization, Mathematical Programming 33 (1985) 68-81.

[7] F. A. Lootsma and K. M. Ragsdell, State-of-the-art in parallel nonlinear optimization, Parallel Computing 6 (1988) 133-155.

[8] Jorge J. Mor and David J. Thuente, Line Search Algorithms with Guaranteed Sufficient Decrease, ACM Transactions on Mathematical Software 20 3 (1994) 286-307.

[9] Bruce A. Murtagh and Michael A. Saunders, Minos 5.4 User's Guide, Technical Report SOL 83-20R, Department of Operations Research, Stanford University, 1983.

[10] I. Pardines, J. J. Pombo and F. F. Rivera, Parallel Algorithm for Backward and Forward Sweeps of Plane Rotations, in: Proc. IASTED International Conference Applied Informatics (Acta Press, Zurich, 2001) 418-423.

[11] I. Pardines and F. F. Rivera, Parallel Quasi-Newton Optimization on Distributed Memory Multiprocessors, in: Parallel Computing, Advances and Current Issues (Imperial College Press, London, 2002) 338-345.

[12] K. H. Phua and R. Setiono, Multi-step, multi-directional parallel algorithms for unconstrained optimization, in: Optimization Techniques and Applications (World Scientific, Singapore, 1992) 481-487.

[13] R. B. Schnabel, Concurrent function evaluations in local and global optimization, Computer Methods in Applied Mechanics and Engineering 64 (1987) 537-552.